Tenth Annual
University of Oregon Programming Competition*

Saturday, May 13, 2006

*We are pleased to acknowledge the support of Google and Microsoft.

BUS STOPS

Lane Transit District is planning a rapid transit bus between Eugene and Springfield that will use various technologies to make the trips faster and more efficient. If there are no passengers to pick up at a station or no current passengers who want to get off, the bus can skip a station along the way. Since skipping too many stations in a row could upset the schedule, a limit is placed on the number of consecutive stations that can be skipped.

Now, it is important that dispatchers be able to anticipate *how many different trip configurations are possible* before the bus leaves the depot, taking into account the destination of the passengers on board at the start of the run. For a simple example, suppose there are only three bus stations between the Eugene and Springfield depots:

If at most 1 station in a row can be skipped, then there are 5 possible trips:

EUG→HIL→UO→GLE→SPR	(no stations skipped)
EUG→UO→GLE→SPR	(just Hilyard skipped)
EUG→HIL→GLE→SPR	(just UO skipped)
EUG→HIL→UO→SPR	(just Glenwood skipped)
EUG→UO→SPR	(Hilyard skipped, Glenwood skipped)

If up to 2 stations in a row can be skipped, then there are 2 additional trips:

EUG→HIL→SPR	(UO and Glenwood skipped)
EUG→GLE→SPR	(Hilyard and UO skipped)

However, *if there is a passenger on board at the depot who is going to get off at UO*, then the 3 trips above that skipped the UO would not be possible, leaving only 4 possible trip configurations.

In this problem, you are to write a program to find *how many* different trips are possible given a particular trip specification.

The first line of the input to your program contains a single integer n indicating the number of trip specifications to follow. Each trip spec is a line of numbers, where the first number is the station count, the second is the skip limit, the third is the number r of required stops, and the remaining r numbers on the line are the numbers of the stations where the bus must stop (i.e., the destinations of the passengers who got on at the starting depot). Stations are numbered beginning with 1; the depots at either end are not included in this numbering.

For each spec, your output should have one line indicating the total number of possible trips for that spec.

Sample Input

```
5
3 1 2 2 3
3 2 1 2
6 0 2 1 2
5 2 0
8 3 3 2 4 5
```

Sample Output

```
2 possible trips
4 possible trips
1 possible trips
24 possible trips
32 possible trips
```

INTERCEPTING SMUGGLERS

The island nation of Bogusnovia has rich deposits of bogusnium, a valuable mineral, but few other resources. Bogusnovia's schools and hospitals are supported entirely by taxes on bogusnium export. With a large number of bogusnium mines in the interior, and ports all along the coast, Bogusnovia also has a problem with smugglers.

Since bogusnium is heavy, significant amounts can only be moved by truck along the road system. As long as every route from mine to port is blocked at some point by a customs inspection post, the smugglers will be thwarted.

All roads run from town to town. Trucks are inspected only upon entry to a town, so a smuggler could sneak a load out of a town with both a mine and inspection post, but a smuggler cannot sneak a load into or through a town with an inspection post.

Your program must determine whether it is possible to deter smuggling.

The first line of the input to your program will contain an integer n indicating the number of problem instances to follow.

Each line of a problem instance begins with `CITY`, `ROAD`, or `END` and is one of the following:

`CITY` *cityname attributes*
`ROAD` *cityname cityname*
`END`

The `ROAD` lines follow the `CITY` lines and indicate direct road links between two previously described cities. A single `END` line indicates the end of the problem instance.

The parts of a line are separated by one or more spaces. A *cityname* is a sequence of alphabetic characters. *Attributes* are a sequence of one or more characters '-', 'P', 'M', or 'I', interpreted as follows:

P	The city is a port.
M	The city has a bogusnium mine.
I	The city has a customs inspection post.
-	None of the above

Thus, a city with attributes `IM` has both a customs inspection post and a bogusnium mine, while a city with attributes – has neither. No port town has a mine.

A smuggling route is any sequence beginning with a mine town, ending with a port town, in which each city in the sequence is connected to the next by a road, and in which no city except possibly the first has an inspection post.

For each problem instance, if there are any smuggling routes in Bogusnovia, your program should print:

```
We are doomed!  The people will have no schools or medicine!
```

If there are no smuggling routes in Bogusnovia, your program should print:

```
Praise President Borisnogun, we have defeated the smugglers!
```

Sample Input

```
2
CITY Maresgrad P
CITY Dullesville I
CITY Podunksburgh -
CITY Dirtpits M
ROAD Maresgrad Dullesville
ROAD Maresgrad Podunksburgh
ROAD Dullesville Dirtpits
ROAD Podunksburgh Dirtpits
END
CITY Blutosgrad  MI
CITY Chloristan  IM
CITY Dargosfurd  M
CITY Intermedsburgh  -
CITY Furthersgrad I
CITY Woolsfurd  PI
CITY Xywxstan   P
ROAD Blutosgrad Intermedsburgh
ROAD Intermedsburgh Furthersgrad
ROAD Chloristan Intermedsburgh
ROAD Dargosfurd Intermedsburgh
ROAD Furthersgrad Woolsfurd
ROAD Furthersgrad Xywxstan
END
```

Sample Output

```
We are doomed!  The people will have no schools or medicine!
Praise President Borisnogun, we have defeated the smugglers!
```

DNA COMBING

An important issue in computational biology is the determination of evolutionary history when genetic material may have been recombined. This problem focuses on a simple model of a one of the possible processes.

Given DNA sequences $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, we want to know whether $\mathcal{C}$ could have been formed from $\mathcal{A}$ and $\mathcal{B}$ by *combing*, a process that involves an interleaving of the two strings. To be precise, you need to determine whether these sequences can be expressed

$$\begin{aligned} \mathcal{A} &= \alpha_1\alpha_2\cdots\alpha_m \\ \mathcal{B} &= \beta_1\beta_2\cdots\beta_m \\ \mathcal{C} &= \alpha_1\beta_1\alpha_2\beta_2\cdots\alpha_m\beta_m \end{aligned}$$

where the α_i and β_i are strings of symbols from the set $\{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}$. (Note: it could be that α_1 and/or β_m is an empty string).

The first line of input will be an integer n indicating the number of problem instances to follow. Each instance will begin with a line with two integers $a\ b$, each ≤ 1000 and separated by a single space, indicating the lengths of strings $\mathcal{A}$ and $\mathcal{B}$, respectively. The strings $\mathcal{A},\mathcal{B},\mathcal{C}$ will follow in order, but since these may be very long, they may have to be split over multiple lines with at most 60 characters per line. Specifically, the next $\lceil\frac{a}{60}\rceil$ lines will contain the string $\mathcal{A}$, with 60 characters in all lines except possibly the last. Following that, $\mathcal{B}$ will be similarly be represented over $\lceil\frac{b}{60}\rceil$ lines. Finally, $\mathcal{C}$ will be represented in the same fashion over $\lceil\frac{a+b}{60}\rceil$ lines.

For each problem instance, you should indicate with the words

`possible` or `impossible`

whether $\mathcal{C}$ could have been formed by a combing of $\mathcal{A}$ and $\mathcal{B}$.

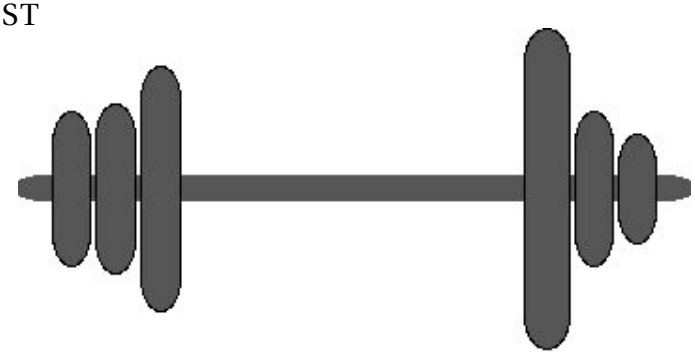
Sample Input

```
7
3 2
AAA
CG
ACAGA
3 2
AAA
CG
CAAAG
3 2
AAA
CG
GAAAC
7 4
AAAGAAA
AAAC
AAAGAAACAAA
5 5
AAGTA
AGTCC
AAGTAGTCAA
7 4
AAAGAAA
AAAC
AAAAAAACGAA
10 10
AGGTCAAGCT
AGGTCAAGTC
AGAGGTCAAGTCAAGTCGCT
```

Sample Output

```
possible
possible
impossible
possible
impossible
impossible
possible
```

WEIGHT LIST



The West Willamette Weightlifting tournament makes use of a fixed set S, of weights that are available for use on a barbell. Having randomized the WWW entrants' positions in a queue, each successive competitor must lift the next subset of S, where the possible subsets are listed in nondecreasing order by total weight. (The weights themselves need not be balanced across the barbell.)

In this problem you are to determine the total weight that would be on the barbell given a weightlifter's position in the queue. That is, you must determine the weight of the t^{th} lightest nonempty subset of S for given S, t. Note that there may be more than one subset with a given total weight, in which case some lucky weightlifters may not have to lift any more weight than their immediate predecessors.

The first line of the input to this problem will indicate the number n of problem instances to follow. The first line of each instance will specify a pair $s\ t$ of integers where s is the number of weights in the set S. The weights in S will be listed, one to a line, over the next s lines. Each weight will be an integer.

For each problem instance $s\ t$, you are to determine the weight that has to be lifted by the t^{th} weightlifter.

Sample Input

```
3
6 5
1
1
1
2
3
3
6 30
6
5
4
3
2
1
5 30
1
2
4
8
16
```

Sample Output

```
Subset 5 has total weight 2.
Subset 30 has total weight 10.
Subset 30 has total weight 29.
```

PSEUDO PRIMES

There are some interesting, but not 100% reliable, methods for testing primality. One of the best-known is inspired by the following special case of *"Fermat's Little Theorem"*.

Theorem. *If p is an odd prime, then p divides $2^{p-1}-1$.*

The converse is not true, but the counterexamples are fairly sparse.

Definition. A composite integer m is called a *Fermat pseudoprime* if m divides $2^{m-1}-1$.[†]

There are only 3 Fermat pseudo-primes less than 1000. The smallest of these is 341.

Less well-known is a test inspired by a remarkable sequence first described by R. Perrin in 1899. The Perrin sequence is recursively defined as follows:

$$\begin{aligned} a_0 &= 3, \\ a_1 &= 0, \\ a_2 &= 2, \\ \text{and, for } m \geq 3, \quad a_m &= a_{m-2} + a_{m-3}. \end{aligned}$$

Perrin proved

Theorem. *If p is prime then p divides a_p.*

Once again, the converse is not true.

Definition. A composite integer m is called a *Perrin pseudoprime* if m divides a_m.

Such m are extremely rare. In fact, none were known until 1982 when Adams and Shanks announced that 271441 is a Perrin pseudoprime. There is only one other Perrin pseudoprime less than a million.

In this problem, you are to test whether given composite integers are pseudoprimes of one or both types. (You are not required to verify compositeness.)

The first line of input to your program will be an integer n indicating how many problem instances will follow, one to a line. Each of the next n lines will have a single composite integer $m < 2^{32}$.

The output, for each test integer m should be one of the following

m `is a Fermat pseudoprime but not a Perrin pseudoprime.`
m `is a Perrin pseudoprime but not a Fermat pseudoprime.`
m `is both a Fermat pseudoprime and a Perrin pseudoprime.`
m `is neither a Fermat pseudoprime nor a Perrin pseudoprime.`

[†] More precisely, such m might be called a Fermat pseudoprime *with respect to base 2*.

Sample Input

```
5
15
341
271441
27664033
46672291
```

Sample Output

```
15 is neither a Fermat pseudoprime nor a Perrin pseudoprime.
341 is a Fermat pseudoprime but not a Perrin pseudoprime.
271441 is a Perrin pseudoprime but not a Fermat pseudoprime.
27664033 is both a Fermat pseudoprime and a Perrin pseudoprime.
46672291 is a Perrin pseudoprime but not a Fermat pseudoprime.
```

PRACTICE PROGRAM

This sample exercise asks you to write a program that adds and multiplies two integers.

Your program should expect input as follows: the first line will contain a single integer n followed by n lines, each of which has a pair of integers separated by a single space.

Each input integer pair should produce one output line that gives the sum and the product of the two integers separated by a single space. Following that should be a blank line and then your output should conclude with the name of your team followed by a blank line followed by the names of your team members, each on its own line.

Sample Input

```
3
2 2
0 3
1 5
```

Sample Output

```
4 4
3 0
6 5

newyorknix

Allan Houston
Stephon Marbury
Kurt Thomas
```